CS1006T Data Strucutres
Unit 2 - List ADT

Dr. V.A.Kandappan

Assistant Professor,
Department of Computer Science,
Shiv Nadar University Chennai.

Lecture	Tutorial	Practical	Credit
3	0	0	3

Unit 2 - List ADT Summary

1. Array Implementation of List
2. Operations on lists (as Arrays)
   - Insertion
   - Deletion
   - Merging
3. Linked lists
   - Singly-linked lists
   - Doubly linked list
   - Circular linked list
   - Operations on linked lists
4. Cursor implementation of lists
5. The polynomial ADT

What is a list?

Definition of list (data structure):

$$\mathcal{L} = \{\mathcal{L}_0, \mathcal{L}_1, \cdots, \mathcal{L}_{N-1}\}$$

- $size(\mathcal{L}) = N$
- If there are no elements in the list, then $size(\mathcal{L}) = 0$
- Successor: $Succ\left(\mathcal{L}_i\right) = \mathcal{L}_{i+1}$
- Predecessor: $Pred\left(\mathcal{L}_i\right) = \mathcal{L}_{i-1}$

What is $Pred\left(\mathcal{L}_0\right)$ and $Succ\left(\mathcal{L}_{N-1}\right)$?

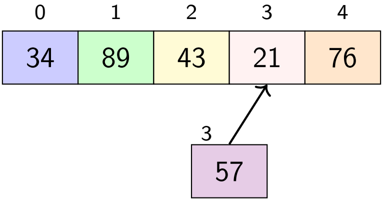
Operations on list

- $PrintList\,(\mathcal{L})$
- $Find\,(\mathcal{L}, key)$
- $InsertBegin\,(\mathcal{L}, \mathcal{L}_{new})$
- $InsertEnd\,(\mathcal{L}, \mathcal{L}_{new})$
- $DeleteElem\,(\mathcal{L}, key)$
- $InsertAtK\,(\mathcal{L}, k, \mathcal{L}_{new})$
- $FindKthElement\,(\mathcal{L}, key)$

Array implementation of List

$$\mathcal{L} = \{\mathcal{L}_0, \mathcal{L}_1, \cdots, \mathcal{L}_{N-1}\}$$

where $\mathcal{L}_i := \mathcal{L}[i]$

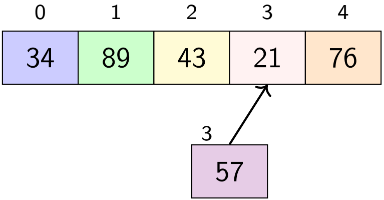


0	1	2	3	4
34	89	43	21	76

3

57

(How do we do?) Operations on List (array implementation)



	Best case Analysis	Worst Case Analysis
PrintList $(\mathcal{L})$		
Find $(\mathcal{L}, key)$		
InsertBegin $(\mathcal{L}, \mathcal{L}_{new})$		
InsertEnd $(\mathcal{L}, \mathcal{L}_{new})$		
DeleteElem $(\mathcal{L}, key)$		
InsertAtK $(\mathcal{L}, k, \mathcal{L}_{new})$		
FindKthElement $(\mathcal{L}, k)$		

Print List

```
function PrintList($\mathcal{L}$)
    $N := size\,(\mathcal{L})$
    for $(i = 0; i < N; i++)$ do
        $Print\,(\mathcal{L}[i])$
    end for
end function
```

Find with key

```
function Find($\mathcal{L}$, $key$)
    $SEARCH\,(\mathcal{L}, key)$
end function
```

Find Kth element

```
function Find($\mathcal{L}$, $k$)
    return $\mathcal{L}[k]$
end function
```

Insert at the begin of the array

```
function InsertBegin($\mathcal{L}, \mathcal{L}_{new}$)
    $N := size\,(\mathcal{L})$
    $REALLOC(\mathcal{L}, N+1)$
    for $(i = N; i > 0; i--)$ do
        $\mathcal{L}[i] = \mathcal{L}[i-1]$
    end for
    $\mathcal{L}[0] = \mathcal{L}_{new}$
end function
```

Insert at the end of the array

```
function InsertEnd($\mathcal{L}, \mathcal{L}_{new}$)
    $N := size\,(\mathcal{L})$
    $REALLOC(\mathcal{L}, N+1)$
    $\mathcal{L}[N+1] := \mathcal{L}_{new}$
end function
```

(How do we do?) Operations on List (array implementation)

Procedure for "Delete Element at beginning, end and k"

Procedure for "Insert element at k^{th} location"

What is the problem with the array implementation of Lists?

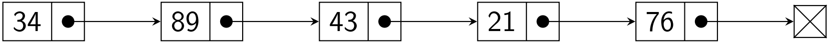
	Best case Analysis	Worst Case Analysis
$PrintList\ (\mathcal{L})$		
$Find\ (\mathcal{L}, key)$		
$InsertBegin\ (\mathcal{L}, \mathcal{L}_{new})$		
$InsertEnd\ (\mathcal{L}, \mathcal{L}_{new})$		
$DeleteElem\ (\mathcal{L}, key)$		
$InsertAtK\ (\mathcal{L}, k, \mathcal{L}_{new})$		
$FindKthElement\ (\mathcal{L}, k)$		

Simple linked list

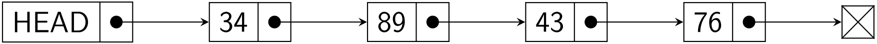
Linked List data structures be like:
I know a guy who knows a guy

Simple linked list

1. Each element has an extra property:"Next"
2. "Next" stores information that helps reach the next element
3. The "Next" of the Last element is "NULL"
4. Address of $Succ\left(L_i\right)$ is stored in "Next"



Simple linked list



	Best case Analysis	Worst Case Analysis
isLast $(\mathcal{L}_H)$		
PrintList $(\mathcal{L}_H)$		
Find $(\mathcal{L}_H, key)$		
InsertBegin $(\mathcal{L}_H, \mathcal{L}_{new})$		
InsertEnd $(\mathcal{L}_H, \mathcal{L}_{new})$		
DeleteElem $(\mathcal{L}_H, key)$		
InsertAtK $(\mathcal{L}_H, k, \mathcal{L}_{new})$		
FindKthElement $(\mathcal{L}_H, k)$		

Find whether the element is last in the list

```
function IsLast($\mathcal{L}_i$)
    if $((\mathcal{L}_i) \to Next == NULL)$ then
        return $True$
    else
        return $False$
    end if
end function
```

Print List

```
function PrintList($\mathcal{L}_i$)
    if $isLast\left(\mathcal{L}_i\right)$ then
        $Print\left(\mathcal{L}_i\right)$
        return
    end if
    $Print\left(\mathcal{L}_i\right)$
    PrintList($Succ\left(\mathcal{L}_i\right)$)
end function
```

Find whether the element is in the list with "*key*"

```
function Find($\mathcal{L}_i, key$)
    if $key\,((\mathcal{L}_i) == key)$ then
        return $True$
    else
        if $isLast\,(\mathcal{L}_i)$ then
            return $False$
        else
            Find($\mathcal{L}_i \rightarrow Next, key$)
        end if
    end if
end function
```

How can we modify the above to delete the element with key?

Insert at beginning

```
function InsertBegin($\mathcal{L}_H, \mathcal{L}_{new}$)
    $\mathcal{L}_{new} \rightarrow Next = \mathcal{L}_H \rightarrow Next$
    $\mathcal{L}_H \rightarrow Next = Info\left(\mathcal{L}_{new} \rightarrow Next\right)$    ▷ Make New element as first and chain the next to new element
end function
```

Insert at end

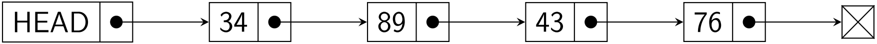
```
function InsertEnd($\mathcal{L}_i, \mathcal{L}_{new}$)
    if $isLast\left(\mathcal{L}_i\right)$ then
        $\mathcal{L}_i \rightarrow Next = Info\left(\mathcal{L}_{new}\right)$
        $\mathcal{L}_{new} \rightarrow Next == NULL$
    else
        InsertEnd($\mathcal{L}_i \rightarrow Next, \mathcal{L}_{new}$)    ▷ Recursive Call
    end if
end function
```

Find the k^{th} element is in the list

```
function FindKthElement($\mathcal{L}_H, k$)
    if $k > N$ then
        print("Length of list is less than $k$")
    else
        $count := 1$
        $\mathcal{L}_{tmp} := Succ\left(\mathcal{L}_H\right)$
        while $count < k - 1$ do
            $\mathcal{L}_{tmp} := Succ\left(\mathcal{L}_{tmp}\right)$
            $count = count + 1$
        end while
        return $\mathcal{L}_{tmp}$
    end if
end function
```

How can we modify the above to insert the element at "k"?

Simple linked list - Asymptotic Analysis



	Best case Analysis	Worst Case Analysis
isLast $(\mathcal{L}_H)$		
PrintList $(\mathcal{L}_H)$		
Find $(\mathcal{L}_H, key)$		
InsertBegin $(\mathcal{L}_H, \mathcal{L}_{new})$		
InsertEnd $(\mathcal{L}_H, \mathcal{L}_{new})$		
DeleteElem $(\mathcal{L}_H, key)$		
InsertAtK $(\mathcal{L}_H, k, \mathcal{L}_{new})$		
FindKthElement $(\mathcal{L}_H, k)$		

In simple linked list or Singly linked list, we have information about $Succ\left(\mathcal{L}_i\right)$ for an i^{th} element.

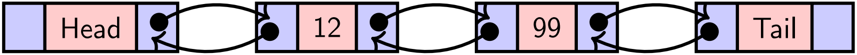
what if your application needs to go back?

(or)

What if you want to reverse the order of elements in the list frequently?

DOUBLY LINKED LISTS BE LIKE

Doubly linked list



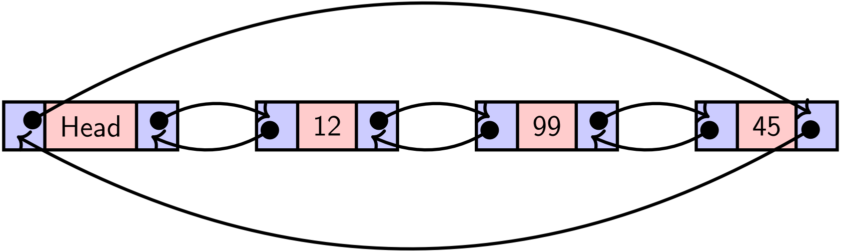
	Best case Analysis	Worst Case Analysis
isLast $(\mathcal{L}_H)$		
PrintList $(\mathcal{L}_H)$		
Find $(\mathcal{L}_H, key)$		
InsertBegin $(\mathcal{L}_H, \mathcal{L}_{new})$		
InsertEnd $(\mathcal{L}_H, \mathcal{L}_{new})$		
DeleteElem $(\mathcal{L}_H, key)$		
InsertAtK $(\mathcal{L}_H, k, \mathcal{L}_{new})$		
FindKthElement $(\mathcal{L}_H, k)$		

circular linked list

Circular linked list

Why and where do we need such type of linked list?

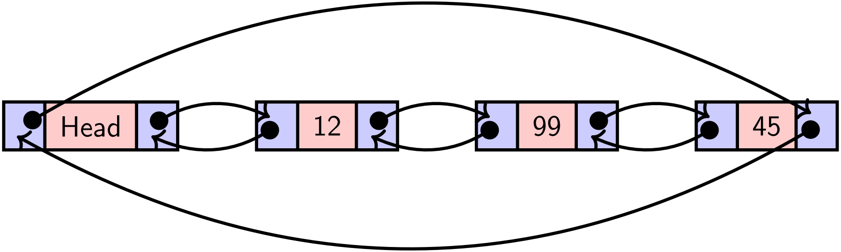
- Any node can be the Head node



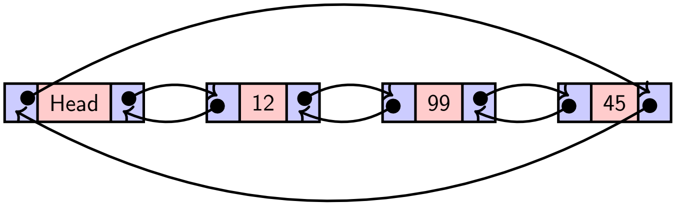
Circular linked list

Why and where do we need such type of linked list?

- Any node can be the Head node
- Round robin, multi-player
- situations where resource allocation is involved

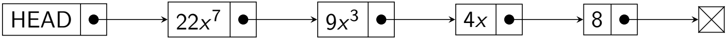


Circular linked list

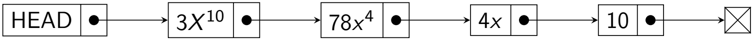


	Best case Analysis	Worst Case Analysis
PrintList $(\mathcal{L}_H)$		
Find $(\mathcal{L}_H, key)$		
InsertBegin $(\mathcal{L}_H, \mathcal{L}_{new})$		
InsertEnd $(\mathcal{L}_H, \mathcal{L}_{new})$		
DeleteElem $(\mathcal{L}_H, key)$		
InsertAtK $(\mathcal{L}_H, k, \mathcal{L}_{new})$		
FindKthElement $(\mathcal{L}_H, k)$		

Example of List - Polynomial ADT



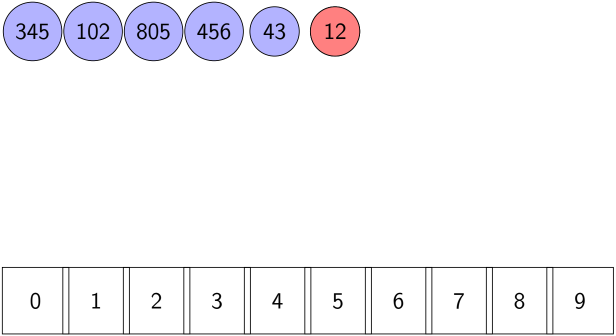
$p_1(x)$



$p_2(x)$

	Best case Analysis	Worst Case Analysis
$ScalePolynial(p_1(x))$		
$AddPolynomials(p_1(x), p_2(x))$		
$MultiplyPolynomials(p_1(x), p_2(x))$		

Example 2 - Radix sort

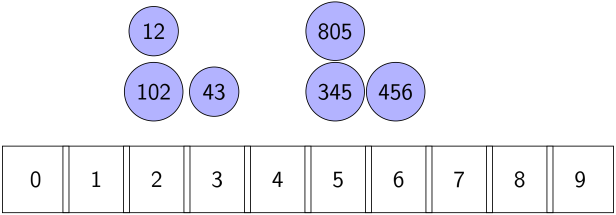


345 102 805 456 43 12

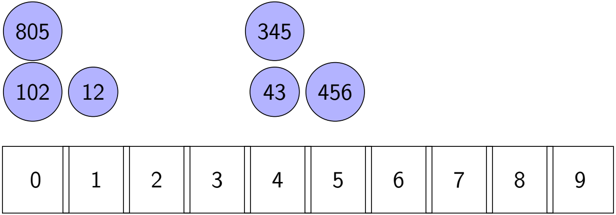
0	1	2	3	4	5	6	7	8	9

Example 2 - Radix sort

- **Step 1**: Sort using “ones” place value

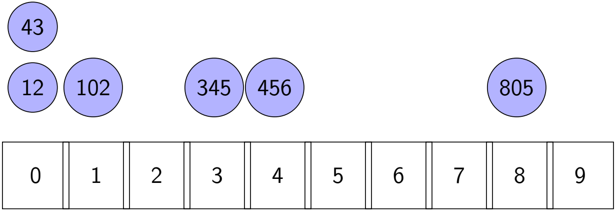


- **Step 2**: Sort using “tens” place value

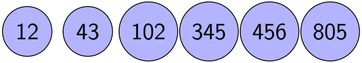


Radix sort (contd.)

- **Step 3**: Sort using “hundreds” place value



- **Sorted Output**:



Is this the fastest sorting algorithm?

We sorted the array in $\mathcal{O}\left(mN\right)$, where m is the number of digits.

Radix Sort using Linked list

1. find the largest element in the array $\mathcal{O}(N)$
2. find the number of digits in the largest element found
3. Initialise 10 linked lists with FetchDeleteAtBegin() and InsertAtEnd() operations, with two arrays *count* and *histcount*
4. For each element in the array, find the "m^{th}" digit and use InsertAtEnd() at the appropriate list.
5. Update two arrays *count* and *histcount*
6. Repeat the following for 2 to m times
    - Use FetchDeleteAtBegin() and update *histcount* and InsertAtEnd() based on the current digit
    - Update two arrays *count* and *histcount*

What C++ offers for Data structures and algorithms?

1. Containers
   - lists
   - vector
   - deque, · · ·
2. Algorithms
3. Iterators
4. Functors

No pointers?

What if your programming language does not support storing addresses as in C/C++?

“Cursor implementation of Lists!!”

Index	Cursor to Next	Value
0	1	-
1	2	-
2	3	-
3	4	-
4	5	-
5	6	-
6	7	-
7	8	-
8	9	-
9	0	-

Cursor implementation of List

List 1:(a, b, c) List2:(z, y, x)

Index	**Cursor to Next**	**Value**
0	5	-
1	2	Head1
2	3	a
3	4	b
4	0	c
5	6	-
6	7	-
7	8	-
8	9	-
9	0	-

Cursor implementation of List

List 1:(a, b, c) List2:(z, y, x)

Index	Cursor to Next	Value
0	9	-
1	2	Head1
2	3	a
3	4	b
4	0	c
5	6	Head2
6	7	z
7	8	y
8	0	x
9	0	-

Data structures lab?